Identifying Cluttering Edges in Near-Planar Graphs

S. van Wageningen[1] and T. Mchedlidze[1] and A. Telea[1]

[1]Utrecht University, The Netherlands

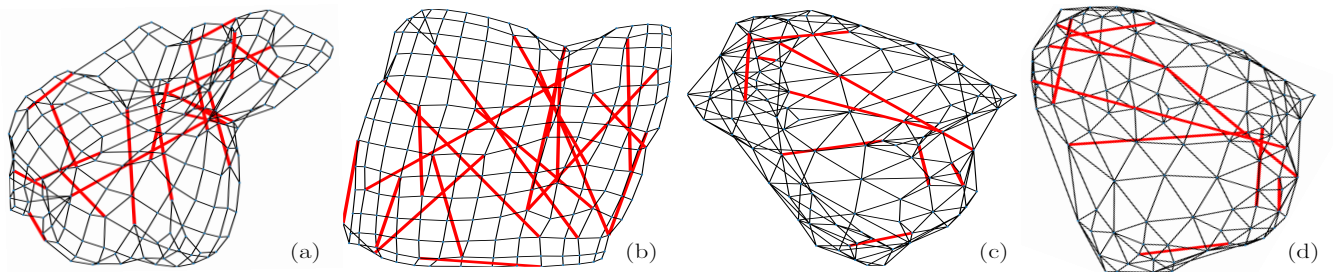

Figure 1: Graphs laid out by ForceAtlas2 with augmenting edges colored red. (a) augmented grid (`redraw`), (b) augmented grid weighted by the `heuristic` ($H_{\min}$), (c) triangulations (`redraw`), (d) triangulations weighted by the `heuristic` ($H_{\min}$)

Abstract

Planar drawings of graphs tend to be favored over non-planar drawings. Testing planarity and creating a planar layout of a planar graph can be done in linear time. However, creating readable drawings of nearly planar graphs remains a challenge. We therefore seek to answer which edges of nearly planar graphs create clutter in their drawings generated by mainstream graph drawing algorithms. We present a heuristic to identify problematic edges in nearly planar graphs and adjust their weights in order to produce higher quality layouts with spring-based drawing algorithms. Our experiments show that our heuristic produces significantly higher quality drawings for augmented grid graphs, augmented triangulations, and deep triangulations.

CCS Concepts

• ***Human-centered computing*** → ***Graph drawings;***

1. Introduction

The ultimate goal when constructing a readable drawing of a graph (i.e. node-link diagram) is to avoid clutter that prevents viewers from grasping the graph's structure. One of the quality metrics that measures the clutter of a graph drawing is the number of edge crossings. It is long known that humans perform better on shortest-path-related tasks in drawings with fewer crossing [Pur97] and tend to prefer such drawings [MBK96, vHR08]. It is then natural to request that a drawing of a graph possesses no edge intersections at all whenever possible. Such drawings, and the graphs that can be drawn in this way, are called *planar*. Detecting whether a graph has a planar drawing [HT74] and constructing one in affirmative [Tam13b] can be done in linear time. However, in practice, graphs are rarely planar. They can still be sparse or contain clear planar substructures – in other words, be *nearly planar*. For such graphs it is desirable to achieve *nearly planar* drawings.

There are various attempts to formalize the notion of *near-planarity* and to construct nearly planar drawings. Unfortunately, all of these attempts lead to hard computational problems [GJ83, CM13, ABS11, KM13]. There are a few spring-based algorithms that address readability issues relevant to near-planarity [ABS12, SAAB11]. Yet, there is a lot of room for further work to design practical layout algorithms for nearly planar graphs.

In this paper we propose a spring-based heuristic approach to construct nearly planar drawings of graphs that contain dense planar substructures. This work is motivated by the lack of comparable approaches and the aforementioned hardness of formal definitions of near-planarity. We conduct an experimental evaluation comparing our approach to state-of-the-art spring-based algorithms. The paper is structured as follows. Section 2 presents initial observations on how to resolve clutter in near planar drawings, and provides an experiment that further motivates our approach. Section 3 describes our heuristic. Section 4 and 5 describe the experimental setup and the results of our experiments.

Related work We denote by $G = (V, E, w)$ a graph, with V and E being the sets of nodes and edges, respectively; and $w : E \to \mathbb{R}$ an edge-weighting function. Here $n = |V|$ and $m = |E|$. For nodes u and v, we denote by $e = \{u, v\}$ and $e = (u, v)$, an undirected edge and a directed edge.

Theoretical approaches to near-planarity Since a planar drawing does not contain any crossing, the most straight-forward idea to define near-planarity is to request a drawing with as few crossings as possible [CM13]. This problem is NP-hard [GJ83] even for a planar graph plus a single additional edge [CM13]. There are a few algorithms that insert edges into planar graphs and their drawings in a crossing-optimal way [RR22, GKM08, GMW05, CH16, CG12] which have been compared experimentally [GM03].

While we know that humans perform tasks well on planar layouts [Pur97], it has been also shown experimentally that the negative effects of the crossings on task performance decrease as the edge crossing angles increase [HHE08]. This led to the definition of RAC [DEL11] and α-AC drawings [GDLM11]. Deciding whether such drawings exist is NP-hard as well [ABS11]. If the graph is sparse, it is also natural to try to limit how many crossings an edge has, as fewer crossings would impair less the perception of that edge. This idea led to definition of *k-planar* and *quasi-planar* drawings. However, again, recognizing whether a graph has these kind of drawings is NP-complete [KM13].

Heuristics A plethora of spring-based algorithms produce high-quality layouts without specifically targeting nearly planar graphs [Tam13a]. Out of these approaches, we next consider ForceAtlas2 (*FA2*) [JVHB14, Chi19] and Stress Majorization (*SM*) [GKN04]. These are powerful layout techniques that we use for our experiments since novel techniques are often compared to them [WJW*19, KCM18]. SM solely bases the spring forces between all pairs of nodes on the length of their shortest paths. FA2 considers attractive and repulsive forces to compute node spring forces. Additionally, the work on SM [GKN04] suggests to weight node pairs by taking into account the number of common neighbors. Here, the weight of a node-pair u, v is set as $w(u,v) = |N_u \cup N_v| - |N_u \cap N_v|$, where N_u denotes the neighborhood of node u. This idea improves the performance of SM when edge lengths need to vary significantly. As will become clear in the following, this approach is relevant to ours and is therefore included as the *neighborhood weighting function* in our experiments.

Finally, relevant to near-planarity, Argyriou et al. [ABS12] present an approach that maximizes the *total resolution*, which is the minimum of the *angular* and *crossing* resolution. In contrast, ImPrEd [SAAB11] preserves the topology of the given layout and therefore its planarity. Similarly, tsNET* [KRM*17] tends to preserve a layout's original structure by favoring occasional long edges and thus unfolding the layout.

2. Preliminaries

Our overarching goal is to produce a drawing of a nearly planar graph G which clearly depicts its planar substructure. This statement itself hints us to distinguish among the graph edges that contribute to its planar substructure and those that destroy it. If we were able to detect the latter edges, we could remove them, construct a planar drawing of the remaining graph (using e.g. [dFPP90, Sch90]), and draw the removed edges atop of it. There are two challenges that prevent us from taking this approach. The first one is that detecting a dense planar substructure is a hard optimization problem known as *Maximum planar subgraph* [Cim95]. The second is that such an approach would inevitably be based on algorithms to construct planar drawings of planar graphs, e.g. [dFPP90, Sch90], which are relatively hard to implement and, also, are not part of most graph drawing libraries and applications. Therefore, we choose to attack our problem using a relatively simple heuristic based on a spring-based approach.

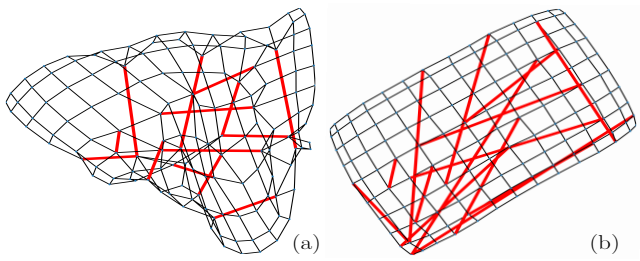

Figure 2: (a) Augmented grid (b), weighted augmented grid

Our initial idea is to identify such *planarity-destroying* edges. Once identified, we can weight them with relatively lower weights than regular edges. We can then use a state-of-the-art spring-based approach, that takes edge-weights into account. Our hope is that the planarity-destroying edges will influence the layout less than the remaining edges and therefore the planar substructure will reveal itself in the drawing.

To test whether this idea is feasible, we perform the following initial experiment. We consider a grid $D = (V, E)$ and construct a graph $G = (V, E \bigcup E_p)$, where E_p is a set of random edges on the vertex set V with the property that $E \bigcap E_p = \emptyset$. We call this graph an *augmented grid*. Starting with a random initial coordinate assignment, we draw an augmented grid using FA2. These layouts (see Figure 2a) appear cluttered and folded inwards, and of poor quality. We then reduce the weights of the edges in E_p to 0.01 and rerun FA2. The resulting layout (Figure 2b) has unfolded and has become a near perfect depiction of a grid graph. This experiment hints that knowing the planarity-destroying edges can be useful in depicting planar substructures in nearly planar graphs by using a spring-based approach and appropriate edge weighting scheme.

In our experiments with other augmented planar graph classes, we observe that not only the planarity-destroying edges create clutter in the drawing, but so do the edges that are close to the outer face. These observations lead to the following two questions: Which edges of a given nearly planar graph G create clutter in the drawings of G generated by a spring-embedding algorithm? Using a spring-based approach, how do we weight these edges to create a drawing where a planar substructure is clearly visible? We call such edges *cluttering* and address the stated questions in the following section. Throughout the paper, we use ♣ to refer the reader to the full version for more details. The Python implementation of the heuristic can be found on GitHub.

3. Finding and weighting cluttering edges

Vertex-disjoint paths Our approach to identify cluttering edges is based on the following intuition. If the end-vertices of an edge $e = \{u, v\}$ are connected by multiple, relatively short paths in the

Dataset	n_{min}	n_{max}	m_{min}	m_{max}	*#Graphs*
Grids	48	196	86	383	50
Triangulations	26	100	69	297	50
Deep triangulations	25	98	86	367	50
Rome	18	100	25	141	100

Table 1: Statistics of datasets used in the experiments

graph, then the edge e can also be short. In the opposite case, where there are only relatively long paths between u and v, then the edge e could collapse the drawing and therefore could be a cluttering one. However, finding all paths between even a single pair of vertices will result in exponential computations. Hence, we use as a proxy the lengths of the vertex-disjoint paths between u and v.

Vertex-disjoint paths can be found using the max-flow Edmonds-Karp algorithm [EK72], as described in [KT06] ♣. In the following, for an edge $e = \{u, v\}$, we denote by $f(e) = [\ell_1, \ell_2, \ldots]$ the sequence of the lengths of the vertex-disjoint paths between u and v in $G \setminus e$, listed in non-decreasing order, and call $f(e)$ the *footprint* of e. Given that the complexity of Edmond-Karp's algorithm is $O(VE^2)$, computation of footprints for all edges of G takes $O(VE^3)$ time.

Outlier detection By analyzing the footprints of augmenting and original edges in augmented grids, we observe differences in the footprints that brings us the idea of using an outlier detection algorithm to detect cluttering edges ♣. We experiment with the *Isolation Forest* technique [LTZ12] which is designed to find anomalous data points in a high-dimensional space ♣.

Footprint normalization and cluttering edge weighting However, before applying the Isolation Forest method to footprints of the edges, we have to ensure that they all have the same length. For this, we expand or contract the footprints, depending on the user-specified number of dimensions k and function $\mathcal{M}$, which can be either the minimum, maximum, or mean function. Equation 1 portrays how the footprints are expanded or contracted, given a footprint $f(e)$ of initial length l and a desired length k.

$$f'(e) = \begin{cases} f(e) \oplus [\mathcal{M}(f(e))]_{k-l} & l < k \\ f(e) & l = k \\ f(e)[0:k-1] \oplus \mathcal{M}(f(e)[k:l]) & l > k \end{cases} \quad (1)$$

In our experiments we evaluate the results for all aforementioned choices of function $\mathcal{M}$, i.e. minimum, maximum, and mean. Based on multiple experiments, we set the weight of edge e to $\mathcal{M}(f(e))$.

4. Experimental setup

Let $G = (V, E)$ be a nearly planar graph. To evaluate our approach, we apply spring-based algorithms to the weighted graph $G_{\mathcal{M}} = (V, E, w)$, $w(e) = \mathcal{M}(f(e))$ and the baseline unweighted graph $G = (V, E)$. We compare the obtained drawings both qualitatively, by exploring them visually, and quantitatively, by using several quality metrics. We next discuss the datasets of our experiment, the way the layouts are computed, and the measured quality metrics.

Datasets We used four types of graphs, as follows (see Table 1).
Grids: We start with a grid with a random number of rows and columns ranging between 6 and 14. Next, we add $0.1n$ edges between random non-adjacent nodes to destroy the grid structure, yielding the *augmented grid*. Note that most, but not every, such added edge introduces edge crossings.

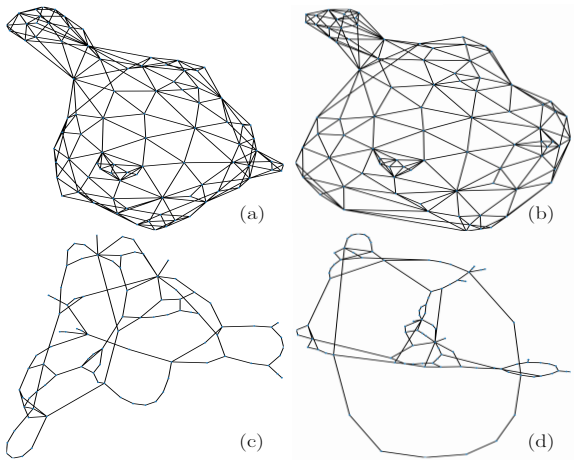

Figure 3: (a) deep triangulations (`orig`), (b) deep triangulations weighted by H_{min}, (c) Rome (`orig`), (d) Rome weighted by H_{min}

Triangulations: These graphs are generated by applying the Delaunay triangulation algorithm on random 2D point sets, to which structure-destroying edges are added similarly to the Grids. We call these graphs *augmented triangulations*. For a grid or a triangulation G, we denote by $\overline{G}$ its augmented version.

Deep triangulations: Even planar graphs can be a challenge for spring-based approaches when it comes to unraveling their planar structure, especially when the planar-layout edges need to have various lengths ♣. To further test our approach, we construct so called *deep triangulations*, as follows: (1) Randomly place $0.7n$ of the vertices and construct their Delaunay triangulation T. (2) Place a random number of points $r < 0.3n$ in a random triangle $t \in T$. (3) Apply Delaunay triangulation to the r points in t to create new edges. (4) Repeat steps 2 and 3 steps until all remaining $0.3n$ vertices have been placed.

Rome: Grids and (deep) triangulations contain dense planar substructures and therefore we expect our heuristic to be able to find cluttering edges in such graphs. We also include a subset of the Rome graphs [GDT] in our experiments, to check whether our technique generalizes to this fairly popular graph benchmark [WYHS21]. Note that these Rome graphs are very sparse and do not contain dense planar substructures.

Layouts We create layouts using the spring-based methods *FA2* and *SM*. For each grid or triangulation G and each spring-method $\mathcal{S}$, we compute seven layouts:

- `orig` $\equiv \mathcal{S}(G)$ – spring-embedding of a graphs G,
- `on_top` – drawing $\mathcal{S}(G)$ with edges of $\overline{G} \setminus G$ appended on top of the drawing, where $\overline{G}$ is the augmented version of graphs G,
- `redraw` $\equiv \mathcal{S}(\overline{G})$ – spring-embedding of $\overline{G}$,
- $H_{\text{min}} \equiv \mathcal{S}(\overline{G}_{\text{min}})$, $H_{\text{max}} \equiv \mathcal{S}(\overline{G}_{\text{max}})$, $H_{\text{mean}} \equiv \mathcal{S}(\overline{G}_{\text{mean}})$ – spring-embeddings of $\overline{G}$ where outlier edges are weighted according to the functions min, max and mean, respectively (see Eqn. 1), jointly referred to as `heuristic` layouts,

- $H_{\text{nb}} \equiv \mathcal{S}(\overline{G}_{\text{nb}})$ – spring-embeddings of $\overline{G}$ where node-pairs are weighted using the neighborhood weighting function.

For the deep triangulations and Rome graphs, which are challenging by themselves, we do not consider augmented versions, thus `on_top` and `redraw` are not computed for them. We run FA2 and SM five times for each graph, and choose the best layout, w.r.t. to the number of crossings. Both SM and FA2 are run for a maximum of 2000 iterations with default settings.

Quality Metrics We quantitatively evaluate the results by computing three quality metrics: the crossing number, `nc` $\in [0, \infty]$ – the total number of crossings in a layout; the angular resolution, `ang_res` $\in [0, 1]$ – the minimum angle between any two incident edges normalized by $2\pi/\max_{v \in V} \deg(v)$; and the crossing resolution, `cros_res` $\in [0, 1]$ – the minimum angle of any two crossing edges normalized by $\pi/2$. Additionally, in order to measure how the augmenting edges distort the layouts, we compute the Procrustes Statistic [CC00], `ps` $\in [0, 1]$. Here, a value of `ps` $= 0$ indicates that two layouts are exactly similar in the positions of vertices, after rotation, translation, and scaling ♣.

5. Results and Discussion

Qualitative analysis Representative examples of running FA2 on augmented graphs (`redraw`) are shown in Figures 1a and 1c ♣. Here, the outer faces appear cluttered and the layouts appear to be folded inwards. Figures 1b and 1d show the layouts where edges are weighted according to the `heuristic`. We observe that the augmenting edges look longer. This makes the layouts being less folded inwards which in turn removes clutter and brings the grid-like and triangulation structures forward. The results of the `heuristic` on deep triangulations (Figure 3a, 3b) also show some clutter decrease, especially in the outer face. Finally, for the Rome graphs (Figures 3c, 3d), we do not see any improvement in the quality of the layout. However, this result is expected, as the Rome graphs do not contain dense plain substructures.

Quantitative analysis Table 2 and the Figure in the full version ♣ show the median values of the quality metrics of layouts of all datasets. Since the data are paired but non-normally distributed, we use the two-sided Wilcoxon signed rank test to indicate significant ($\alpha = 0.05$) improvement or deterioration in a quality metric. For the grids and triangulations we measure whether there is a significant difference between the `heuristic` and the `redraw` layouts. Whereas, for deep triangulations and Rome graphs we compare the `heuristic` with the `orig` layout.

Regarding the number of crossings, we observe significant improvements on the `heuristic` layouts of the augmented grids and (deep) triangulation graphs. Additionally, the `heuristic` approach outshines the neighborhood technique for these graphs. As expected from the qualitative analysis, for the Rome graphs the `heuristic` layouts are either equally good (H_{max}) or much worse (H_{mean}, H_{min}) concerning the number of crossings. The full version of Table 2 also contains the results of (H_{mean}) and (H_{min}) ♣.

Regarding angular resolution, we see significant improvements over the `redraw` layouts in the augmented grids when FA2 is used. Also, SM scores significantly worse for both the triangulations and Rome graphs. For the crossing resolution, we observe no significant differences for the `heuristic` for the augmented grids, triangulations, and deep triangulations. However, the `heuristic` layouts are significantly worse for the Rome graphs, as expected from the structure of the Rome graphs and the intention of our heuristic.

Grids	nc		ang_res		cros_res		ps	
orig	0	0	.631	.982	-	-	-	-
on_top	52	52	.011	.005	.10	.16	0	0
redraw	67	102	.014	.018	.14	.12	.11	.25
H_{max}	51	52	.028	.020	.14	.14	.02	.02
H_{nb}	60	94	.019	.026	.17	.12	.11	.19
Triangulations								
orig	53	79	.009	.023	.10	.14	-	-
on_top	96	131	.008	.016	.08	.10	0	0
redraw	87	154	.009	.019	.12	.12	.03	.08
H_{max}	65	88	.013	.012	.12	.12	.02	.06
H_{nb}	93	124	.013	.015	.10	.09	.04	.07
Deep triangulations								
orig	74	102	.017	.020	.13	.15	-	-
H_{max}	58	82	.016	.022	.16	.13	.03	.09
H_{nb}	73	91	.017	.023	.10	.13	.03	.04
Rome								
orig	26	30	.048	.065	.30	.26	-	-
H_{max}	24	34	.037	.029	.21	.15	.35	.53
H_{nb}	25	25	.052	.062	.27	.29	.13	.18

Table 2: Median values of metrics. For grids and triangulations, `redraw` is compared with `heuristic` layouts. For deep triangulations and Rome, `orig` is compared with `heuristic` layouts. A stronger hue indicates a significant result, with FA2 & SM

Additionally, we note that the Procrustes Statistic values for the `heuristic` layouts tend to be close to 0 for all but the Rome graphs. These results indicate that the weighting tactic of our heuristic yields layouts that stay close to the original planar structure. Lastly, we observe that FA2 scores better than SM on most metrics for all datasets.

6. Conclusion

We presented a heuristic to detect edges that create clutter in layouts of near planar graphs. By suitably weighting such edges, we use spring-embedders to draw these graphs with the goal to better convey their planar substructures. The experiments indicate that our heuristic produces better results for augmented grids and triangulations. For deep triangulations we noticed visual improvements and clutter decrease in the outer face, though further improvements are possible. Moreover, our heuristic produces drawings with fewer number of crossings than conventional methods for all but the Rome graphs. This result is, however, expected since the Rome graphs do not contain dense planar substructures. Future work can yield more insight into deep triangulations, which we expect to be very challenging to lay out in a way that reveal their planar structure. Moreover, additional comparisons can be made between our heuristic and tsNET*. In addition to more experiments, future work can attempt to improve the heuristic's limiting time complexity, by altering or substituting the vertex-disjoint path and outlier detection computations. Finally, we plan to test whether Graph Neural Networks can be more successful in identifying cluttering edges.

References

[ABS11] ARGYRIOU E. N., BEKOS M. A., SYMVONIS A.: The straight line rac drawing problem is np-hard. In *SOFSEM 2011: Theory and Practice of Computer Science* (Berlin, Heidelberg, Germany, 2011), vol. 6543 of *LNCS*, Springer, pp. 77–85. doi:10.1007/978-3-642-18381-2_6. 1, 2

[ABS12] ARGYRIOU E., BEKOS M., SYMVONIS A.: Maximizing the Total Resolution of Graphs. *The Computer Journal 56*, 7 (06 2012), 887–900. doi:10.1093/comjnl/bxs088. 1, 2

[CC00] COX T., COX M. A.: *Multidimensional scaling*. Chapman and Hall/CRC, 2000. 4

[CG12] CHIMANI M., GUTWENGER C.: Advances in the planarization method: Effective multiple edge insertions. *Journal of Graph Algorithms and Applications 16*, 3 (2012), 729–757. doi:10.7155/jgaa.00264. 2

[CH16] CHIMANI M., HLINENÝ P.: Inserting multiple edges into a planar graph. In *32nd International Symposium on Computational Geometry* (Boston, MA, USA, 2016), Fekete S. P., Lubiw A., (Eds.), vol. 51 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 30:1–30:15. doi:10.4230/LIPIcs.SoCG.2016.30. 2

[Chi19] CHIPPADA B.: Forceatlas2 python. URL: https://github.com/bhargavchippada/forceatlas2. 2

[Cim95] CIMIKOWSKI R. J.: An analysis of some heuristics for the maximum planar subgraph problem. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, California, USA, 1995), Clarkson K. L., (Ed.), ACM/SIAM, pp. 322–331. URL: http://dl.acm.org/citation.cfm?id=313651.313713. 2

[CM13] CABELLO S., MOHAR B.: Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM Journal on Computing 42*, 5 (2013), 1803–1829. doi:10.1137/120872310. 1, 2

[DEL11] DIDIMO W., EADES P., LIOTTA G.: Drawing graphs with right angle crossings. *Theoretical Computer Science 412* (2011), 5156–5166. 2

[dFPP90] DE FRAYSSEIX H., PACH J., POLLACK R.: How to draw a planar graph on a grid. *Comb. 10*, 1 (1990), 41–51. doi:10.1007/BF02122694. 2

[EK72] EDMONDS J., KARP R. M.: Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM 19* (1972), 248–264. doi:10.1145/321694.321699. 3

[GDLM11] GIACOMO E. D., DIDIMO W., LIOTTA G., MEIJER H.: Area, curve complexity, and crossing resolution of non-planar graph drawings. *Theory Comput. Syst. 49*, 3 (2011), 565–575. doi:10.1007/s00224-010-9275-6. 2

[GDT] GDTOOLKIT: Rome graphs. URL: http://www.graphdrawing.org/data.html. 3

[GJ83] GAREY M. R., JOHNSON D. S.: Crossing number is np-complete. *SIAM Journal on Algebraic and Discrete Methods 4* (1983), 312–316. 1, 2

[GKM08] GUTWENGER C., KLEIN K., MUTZEL P.: Planarity testing and optimal edge insertion with embedding constraints. *J. Graph Algorithms Appl. 12*, 1 (2008), 73–95. doi:10.7155/jgaa.00160. 2

[GKN04] GANSNER E. R., KOREN Y., NORTH S.: Drawing by stress majorization. In *Graph Drawing* (Berlin, Heidelberg, Germany, 2004), Pach J., (Ed.), vol. 3383 of *LNCS*, SPRINGER. doi:10.1007/978-3-540-31843-9_25. 2

[GM03] GUTWENGER C., MUTZEL P.: An experimental study of crossing minimization heuristics. In *Graph Drawing* (Perugia, Italy, 2003), vol. 2912 of *LNCS*, Springer, pp. 13–24. doi:10.1007/978-3-540-24595-7_2. 2

[GMW05] GUTWENGER C., MUTZEL P., WEISKIRCHER R.: Inserting an edge into a planar graph. *Algorithmica 41*, 4 (2005), 289–308. doi:10.1007/s00453-004-1128-8. 2

[HHE08] HUANG W., HONG S.-H., EADES P.: Effects of crossing angles. In *2008 IEEE Pacific Visualization Symposium* (2008), pp. 41–46. doi:10.1109/PACIFICVIS.2008.4475457. 2

[HT74] HOPCROFT J., TARJAN R. E.: Efficient planarity testing. *Journal of the Association for Computing Machinery 21* (1974), 549–568. doi:10.1145/321850.321852. 1

[JVHB14] JACOMY M., VENTURINI T., HEYMANN S., BASTIAN M.: Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PLOS ONE 9* (2014). doi:10.1371/journal.pone.0098679. 2

[KCM18] KWON O.-H., CRNOVRSANIN T., MA K.-L.: What would a graph look like in this layout? a machine learning approach to large graph visualization. *IEEE Transactions on Visualization and Computer Graphics 24* (2018), 477–488. doi:10.1109/TVCG.2017.2743858. 2

[KM13] KORZHIK V. P., MOHAR B.: Minimal obstructions for 1-immersions and hardness of 1-planarity. *Journal of Graph Theory 72* (2013), 30–71. 1, 2

[KRM*17] KRUIGER J., RAUBER P., MARTINS R., KERREN A., KOBOUROV S., TELEA A.: Graph layouts by t-sne. *Computer Graphics Forum 36* (06 2017), 283–294. doi:10.1111/cgf.13187. 2

[KT06] KLEINBERG J. M., TARDOS É.: *Algorithm design*. Addison-Wesley, 2006. 3

[LTZ12] LIU F. T., TING K. M., ZHOU Z.-H.: Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data 6*, 1 (2012). doi:10.1145/2133360.2133363. 3

[MBK96] MCGRATH C., BLYTHE J., KRACKHARDT D.: Seeing groups in graph layout. *Connections 19* (1996), 22–29. 1

[Pur97] PURCHASE H.: Which aesthetic has the greatest effect on human understanding? In *Graph Drawing* (Berlin, Heidelberg, Germany, 1997), Dibattista G., (Ed.), vol. 1353 of *LNCS*, Springer, pp. 248–261. doi:10.1007/3-540-63938-1_67. 1, 2

[RR22] RADERMACHER M., RUTTER I.: Inserting an edge into a geometric embedding. *Computational Geometry 102* (2022), 101843. doi:10.1016/j.comgeo.2021.101843. 2

[SAAB11] SIMONETTO P., ARCHAMBAULT D., AUBER D., BOURQUI R.: Impred: An improved force-directed algorithm that prevents nodes from crossing edges. *Comput. Graph. Forum 30*, 3 (2011), 1071–1080. doi:10.1111/j.1467-8659.2011.01956.x. 1, 2

[Sch90] SCHNYDER W.: Embedding planar graphs on the grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, California, USA, 1990), Johnson D. S., (Ed.), SIAM, pp. 138–148. URL: http://dl.acm.org/citation.cfm?id=320176.320191. 2

[Tam13a] TAMASSIA R.: *Force-Directed Drawing Algorithms*. Chapman and Hall/CRC, 2013. 2

[Tam13b] TAMASSIA R.: *Planar straight line drawing algorithms*. Chapman and Hall/CRC, 2013. 1

[vHR08] VAN HAM F., ROGOWITZ B.: Perceptual organization in user-generated graph layouts. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1333–1339. doi:10.1109/TVCG.2008.155. 1

[WJW*19] WANG Y., JIN Z., WANG Q., CUI W., MA T., QU H.: Deepdrawing: A deep learning approach to graph drawing. In *IEEE Transactions on Visualization and Computer Graphics* (2019). doi:10.48550/ARXIV.1907.11040. 2

[WYHS21] WANG X., YEN K., HU Y., SHEN H.: Deepgd: A deep learning framework for graph drawing using gnn. *IEEE Computer Graphics and Applications 41*, 05 (2021), 32–44. doi:10.1109/MCG.2021.3093908. 3